Expert "Separate" Build Instructions

Check for platform−specific instructions above and set the environment variable CC if needed.

Note: On some platforms you may need to set the CC environment variable before building.

CDAT can be built in three separate phases:

1. **PYSRC –** To install Python with tcl/tk:

```
cd <CDAT_SRC_DIRECTROY>/pysrc
./install_script <CDAT_INSTALL_DIRECTORY>   # To skip this step, see PYSRC below
```

2. **EXSRC –** To install the external software required by CDAT:

```
cd <CDAT_SRC_DIRECTORY>/exsrc
./install_script <CDAT_INSTALL_DIRECTORY>    # infrequently changing components
```

3. **CDAT –** To install CDAT itself:

```
# installs CDAT and user-contributed packages
# contrib can be turned off with the option "--without-contrib"
<CDAT_INSTALL_DIRECTORY>/bin/python install.py [options]
```

PYSRC: Install Python (if necessary)

The "pysrc" distribution contains everything you need to run CDAT: zlib, readline, tcl, tk, and Python.

The source code versions distributed are:

- Python 2.4.3
- Tcl/Tk 8.4.13
- readline 5.1
- zlib 1.1.4

Reasons you might build a new python include:

- You do not have Python installed in a directory where you have write permission.
- You get a message during CDAT installation saying that your Python is too old.
- You do not want CDAT installed into your existing Python.
- You want to avoid issues such as having the right Python but the wrong Tcl/Tk.

Everything needed for Python, including Tcl and Tk, are included in this package. If in doubt, please compare the versions of the tar files in `<CDAT_SRC_DIRECTORY>/pysrc/src` to be sure your version numbers are as recent as ours.

Obtain the pysrc distribution and unpack it. In subdirectory `<CDAT_SRC_DIRECTORY>/pysrc`, type:

```
./install_script [--debug] <CDAT_INSTALL_DIRECTORY>
```

The build takes place in subdirectory "build", and you'll find log files there.

You can clean up the build directories by issuing the command:

```
cd <CDAT_SRC_DIRECTORY>/pysrc
./clean_script
```

EXSRC: Install the "external" software (if necessary)

Our "exsrc" distribution contains additional required packages. These packages, except for Pyfort, were not written at PCMDI. We have separated them into their own distribution to reduce the amount of software the users must download and rebuild when new versions of CDAT are released and these parts haven't changed.

To build exsrc, type:

```
cd <CDAT_SRC_DIRECTORY>/exsrc
./install_script [--debug] <CDAT_INSTALL_DIRECTORY>
```

The build takes place in subdirectory "build," and you'll find log files there.

You can clean up the build directories by issuing the command:

```
cd <CDAT_SRC_DIRECTORY>/exsrc
./clean_script
```

You only need to do this step once. Minor CDAT updates will not require you to download and build this part again.

If you need to rebuild only some parts of it, you can enable/disable any PACKAGE by adding the `--with-PACKAGE` or `--without-PACKAGE`, or build a single PACKAGE by using `--PACKAGE-only`. To view the list of external software packages, type:

```
cd <CDAT_SRC_DIRECTORY>/exsrc
./install_script . --help
```

The Packages available are:

- NetCDF 3.6
- OpenDAP 3.5 (either that or NetCDF, to turn off −−disable−opendap): libdap 3.5.3, libnc−dap 3.5.2
- Pyfort 8.5.1
- Numeric 23.1
- XGKS 2.6.1
- Pmw 1.2
- Ghostscript 8.50 with jpeg 6b and libpng 1.2.8
- gplot 1989
- gifsicle 1.35
- pbmplus 1991 (Distributed but not used for Linux/Mac)
- netpbm 10.27 (Replaces pbmplus on Linux and Mac system)
- gifmerge 1999
- R 2.2.0 (not installed by default, you'll need to pass −−enable−r)
- VTK 4.2.3 (not installed by default, you'll need to pass −−enable−vtk)
- cmake 1.8.3 (installed by VTK)
- ioapiÂ Â Â Â 3.0.2Â (not installed by default, turns off openDAP, use −−enable−ioapi)
- gdalÂ Â Â Â Â 1.3.0Â (installed with ioapi)

CDAT: Installing CDAT itself

SETUP configuration file (optional):

CDAT installation can be controlled with a configuration file. The installation begins by reading the file installation/standard.py. Options you can set are detailed in that file.

You can use an alternate configuration file (say myconfig.py) by using in step 1:

- `--configuration=myconfig`

Example of a configuration file: The file `installation/pcmdi.py` is an example of a configuration file, in this case the configuration we use at PCMDI. Once a configuration has been specified, it will not be redone in subsequent installs unless you use the `--force` option.

Additional optional steps

Note that in creating your own configuration file, you can use the full power of Python.

A "control file" also governs the install process. The standard control file is `installation/control.py`; `installation/debug.py` is used for the debug builds. You shouldn't have to change anything in these files.

Running "install.py"

```
<CDAT_INSTALL_DIRECTORY>/bin/python install.py [options] [control-files]
```

In the following step you must execute the Python from which you intend to install CDAT. Either use a full path name or make sure that the desired python is the one being chosen by your path. To emphasize this we'll show the command as a full pathname below. Please read notes below before building.

A full usage description is given in file HELP.txt. It can also be viewed with:

```
<CDAT_INSTALL_DIRECTORY>/bin/python install.py --help
```

Standard install (now includes "contrib" Packages):

```
<CDAT_INSTALL_DIRECTORY>/bin/python install.py
```

Standard without contributed packages:

```
<CDAT_INSTALL_DIRECTORY>/bin/python install.py --without-contrib
```

To install after changing the configuration, downloading a new version, or changing platforms, use the `--force option`. For example:

```
<CDAT_INSTALL_DIRECTORY>/bin/python install.py --force
```

To install using your own configuration file:

```
# myconfig is your configuration file name
<CDAT_INSTALL_DIRECTORY>/bin/python install.py --configuration=myconfig
```

A list of all packages installed will appear. If a package fails, an error message will be generated at the end of the install. (All failed Packages will be listed.)

If the build has failed for some packages, you should then examine the log file for the software component it was working on when it failed. For comparison, the subdirectory "logs/samples" contains the log files for a build on all platforms.

A certain class of errors, such as a failure to locate the X11 libraries, can result in the build halting in an inconsistent state. Typically you will fix such a problem by adding a configuration or control file and rebuilding. Be sure to use the `--force` option when you rebuild to insure a complete rebuild.

The set of packages to build is determined by the "packages." To modify the list of packages, make a control file that specifies the desired variable list.

Many of the contributed packages in subdirectory "contrib" require a Fortran compiler. If the default compiler chosen by Pyfort is not what you want, you may need to edit file `configuration.py` in the Pyfort source and reinstall it. See http://pyfortran.sf.net.